Lab Manual

for

Linear Algebra

by

Jim Hefferon

Cover: my Chocolate Lab, Suzy.

Contents

Vector Spaces

Sage can operate with vector spaces, for example by finding a basis for a space.

In this chapter vector spaces take scalars from the reals numbers $\mathbb{R}$ (the book's final chapter uses the complex numbers). You can choose from two models of the real numbers. One is `RDF`, the computer's built-in floating point model of real numbers.[1] The other is `RR`, an arbitrary precision model of reals.[2] The second model is useful in some circumstances but the first runs faster and is more widely used in practice so that is what we will use.

Real n-spaces

Start by creating a real vector space $\mathbb{R}^3$.

```
sage: V = RDF^3
sage: V
Vector space of dimension 3 over Real Double Field
```

You can test for membership.

```
sage: v1 = vector(RDF, [1, 2, 3])
sage: v1
(1.0, 2.0, 3.0)
sage: v1 in V
True
sage: v2 = vector(RDF, [1, 2, 3, 4])
sage: v2 in V
True
```

Next we can try subspaces. The plane through the origin in $\mathbb{R}^3$ described by the equation $x - 2y + 2z = 0$ is a subspace of $\mathbb{R}^3$. You can describe it as a span.

$$W = \{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \bigm| x = 2y - 2z \} = \{ \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} y + \begin{pmatrix} -2 \\ 0 \\ 1 \end{pmatrix} z \bigm| y, z \in \mathbb{R} \}$$

You can create that subspace and you can check it by doing some membership tests.

```
sage: V = RDF^3
sage: v1 = vector(RDF, [2, 1, 0])
sage: v2 = vector(RDF, [-2, 0, 1])
```

[1]This computer uses IEEE 754 double-precision binary floating-point numbers; if you have programmed then you may know this number model as binary64. [2]It defaults to 53 bits of precision to reproduce double-precision computations.

```
sage: W = V.span([v1, v2])
sage: v3 = vector(RDF, [0, 1, 1])
sage: v3 in W
True
sage: v4 = vector(RDF, [1, 0, 0])
sage: v4 in W
False
```

For a closer look at what's happening here, create a subspace of $\mathbb{R}^4$.

```
sage: V = RDF^4
sage: V
Vector space of dimension 4 over Real Double Field
sage: v1 = vector(RR, [2, 0, -1, 0])
sage: W = V.span([v1])
sage: W
Vector space of degree 4 and dimension 1 over Real Double Field
Basis matrix:
[ 1.0  0.0 -0.5  0.0]
```

Sage has identified the dimension of the subspace and found a basis containing one vector (it prefers the vector with a leading 1).

As earlier, the membership set relation works here.

```
sage: v2 = vector(RR, [2, 1, -1, 0])
sage: v2 in W
False
sage: v3 = vector(RR, [-4, 0, 2, 0])
sage: v3 in W
True
```

Basis Sage has a command to retrieve a basis for a space.

```
sage: V = RDF^2
sage: v = vector(RDF, [1, -1])
sage: W = V.span([v])
sage: W.basis()
[
(1.0, -1.0)
]
```

This is the basis you will see if you ask for a description of W.

```
sage: W
Vector space of degree 2 and dimension 1 over Real Double Field
Basis matrix:
[ 1.0 -1.0]
```

Of course, there are subspaces with bases of size greater than one.

```
sage: V = RDF^3
sage: v1 = vector(RDF, [1, -1, 0])
sage: v2 = vector(RDF, [1, 1, 0])
sage: W = V.span([v1, v2])
sage: W.basis()
```

```
[
(1.0, 0.0, 0.0),
(0.0, 1.0, 0.0)
]
```

Adding a linearly dependent vector $\vec{v}_3$ to the spanning set doesn't change the space.

```
sage: W = V.span([v1, v2])
sage: v3 = vector(RDF, [2, 3, 0])
sage: W_prime = V.span([v1, v2, v3])
sage: W_prime.basis()
[
(1.0, 0.0, 0.0),
(0.0, 1.0, 0.0)
]
```

In the prior example, notice that Sage does not simply give you back the linearly independent vectors that you gave it. Instead, by default Sage takes the vectors from the spanning set as the rows of a matrix, brings that matrix to reduced echelon form, and reports the nonzero rows as the members of the basis. Because each matrix has one and only one reduced echelon form, each vector subspace of real n-space has one and only one such basis; this is the *canonical basis* for the space.

It is this basis that Sage shows when you ask for a description of the space.

```
sage: W = V.span([v1, v2])
sage: W
Vector space of degree 3 and dimension 2 over Real Double Field
Basis matrix:
[1.0 0.0 0.0]
[0.0 1.0 0.0]
sage: v3 = vector(RDF, [2, 3, 0])
sage: W_prime = V.span([v1, v2, v3])
sage: W_prime
Vector space of degree 3 and dimension 2 over Real Double Field
Basis matrix:
[1.0 0.0 0.0]
[0.0 1.0 0.0]
```

If you are keen on your own basis then Sage will accomodate you.

```
sage: V = RDF^3
sage: v1 = vector(RDF, [1, 2, 3])
sage: v2 = vector(RDF, [2, 1, 3])
sage: W = V.span_of_basis([v1, v2])
sage: W.basis()
[
(1.0, 2.0, 3.0),
(2.0, 1.0, 3.0)
]
sage: W
Vector space of degree 3 and dimension 2 over Real Double Field
User basis matrix:
[1.0 2.0 3.0]
[2.0 1.0 3.0]
```

Equality You can test whether spaces are equal.

```
sage: V = RDF^4
sage: v1 = vector(RDF, [1, 0, 0, 0])
sage: v2 = vector(RDF, [1, 1, 0, 0])
sage: W12 = V.span([v1, v2])
sage: v3 = vector(RDF, [2, 1, 0, 0])
sage: W13 = V.span([v1, v3])
```

One way to assure yourself that the two spaces $W_{1,2}$ and $W_{1,3}$ are equal is to use set membership.

```
sage: v3 in W12
True
sage: v2 in W13
True
```

Then, since obviously $\vec{v}_1 \in W_{1,2}$ and $\vec{v}_1 \in W_{1,3}$, the two spans are equal.

But the more straightforward approach is to just ask Sage.

```
sage: W12 == W13
True
```

Your exercise of == equality between spaces would be only half-hearted without an test of inequality.

```
sage: v4 = vector(RDF, [1, 1, 1, 1])
sage: W14 = V.span([v1, v4])
sage: v2 in W14
False
sage: v3 in W14
False
sage: v4 in W12
False
sage: v4 in W13
False
sage: W12 == W14
False
sage: W13 == W14
False
```

This illustrates a point about algorithms. Sage could check for equality of two spans by checking whether every member of the first spanning set is in the second space and vice versa (since the two spanning sets are finite). But Sage does something different. For each space it maintains the canonical basis and it checks for equality of spaces just by checking whether they have the same canonical bases. These two algorithms have the same external behavior, in that both decide whether two spaces are equal. But the algorithms differ internally, and as a result the second is faster. Finding the fastest way to do jobs is an important research area of computing.

Operations Sage finds the intersection of two spaces. Consider these members of $\mathbb{R}^3$.

$$\vec{v}_1 = \begin{pmatrix}1\\0\\0\end{pmatrix} \quad \vec{v}_2 = \begin{pmatrix}0\\1\\0\end{pmatrix} \quad \vec{v}_3 = \begin{pmatrix}0\\0\\3\end{pmatrix}$$

Form two spans, the xy-plane $W_{1,2} = [\vec{v}_1, \vec{v}_2]$ and the yz-plane $W_{2,3} = [\vec{v}_2, \vec{v}_3]$. The intersection of these two is the y-axis.

```
sage: V=RDF^3
sage: v1 = vector(RDF, [1, 0, 0])
sage: v2 =  vector(RDF, [0, 1, 0])
sage: W12 = V.span([v1, v2])
sage: W12.basis()
[
(1.0, 0.0, 0.0),
(0.0, 1.0, 0.0)
]
sage: v3 = vector(RDF, [0, 0, 2])
sage: W23 = V.span([v2, v3])
sage: W23.basis()
[
(0.0, 1.0, 0.0),
(0.0, 0.0, 1.0)
]
sage: W = W12.intersection(W23)
sage: W.basis()
[
(0.0, 1.0, 0.0)
]
```

Remember that the trivial space $\{\vec{0}\}$ is the span of the empty set.

```
sage: v3 = vector(RDF, [1, 1, 1])
sage: W3 = V.span([v3])
sage: W3.basis()
[
(1.0, 1.0, 1.0)
]
sage: W4 = W12.intersection(W3)
sage: W4.basis()
[

]
```

Sage will also find the sum of spaces, the span of their union.

```
sage: W5 = W12 + W3
sage: W5.basis()
[
(1.0, 0.0, 0.0),
(0.0, 1.0, 0.0),
(0.0, 0.0, 1.0)
]
sage: W5 == V
True
sage: W5
Vector space of degree 3 and dimension 3 over Real Double Field
Basis matrix:
[1.0 0.0 0.0]
```

```
[0.0 1.0 0.0]
[0.0 0.0 1.0]
```

Other spaces

These computations extend to vector spaces that aren't a subspace of some $\mathbb{R}^n$. You only need to find a real space that is just like the one you have.

Consider the vector space of quadratic polynomials (under the usual operations of polynomial addition and scalar multiplication).

$$\{a_2x^2 + a_1x + a_0 \mid a_2 - a_1 = a_0\} = \{(a_1 + a_0)x^2 + a_1x + a_0 \mid a_1, a_0 \in \mathbb{R}\}$$

It is just like this subspace of $\mathbb{R}^3$.[1]

$$\{\begin{pmatrix} a_1 + a_0 \\ a_1 \\ a_0 \end{pmatrix} \mid a_1, a_0 \in \mathbb{R}\} = \{\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} a_1 + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} a_0 \mid a_1, a_0 \in \mathbb{R}\}$$

```
sage: V = RDF^3
sage: v1 = vector(RDF, [1, 1, 0])
sage: v2 = vector(RDF, [1, 0, 1])
sage: W = V.span([v1, v2])
sage: W.basis()
[
(1.0, 0.0, 1.0),
(0.0, 1.0, -1.0)
]
```

Similarly you can represent this space of 2×2 matrices

$$\{\begin{pmatrix} a & b \\ c & d \end{pmatrix} \mid a - b + c = 0 \text{ and } b + d = 0\}$$

by finding a real n-space just like it. Rewrite the given space

$$\{\begin{pmatrix} a & b \\ c & d \end{pmatrix} \mid a = -c - d \text{ and } b = -d\} = \{\begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix} c + \begin{pmatrix} -1 & -1 \\ 0 & 1 \end{pmatrix} d \mid c, d \in \mathbb{R}\}$$

and then here is a natural matching real space.

```
sage: V = RDF^4
sage: v1 = vector(RDF, [-1, 0, 1, 0])
sage: v2 = vector(RDF, [-1, -1, 0, 1])
sage: W = V.span([v1, v2])
sage: W.basis()
[
(1.0, -0.0, -1.0, -0.0),
(0.0, 1.0, 1.0, -1.0)
]
```

[1]The textbook's chapter on Maps Between Spaces makes "just like" precise.

You could have gotten another matching space by going down the columns instead of across the rows.

```
sage: V = RDF^4
sage: v1 = vector(RDF, [-1, 1, 0, 0])
sage: v2 = vector(RDF, [-1, 0, -1, 1])
sage: W = V.span([v1, v2])
sage: W.basis()
[
(1.0, 0.0, 1.0, -1.0),
(0.0, 1.0, 1.0, -1.0)
]
```

There are other ways to produce a matching space. They look different than each other but the important things about the spaces, such as dimension, are unaffected by their look. That is the subject of the book's third chapter.

Bibliography

Robert A. Beezer. Sage for Linear Algebra. http://linear.ups.edu/download/fcla-2.22-sage-4.7.1-preview.pdf, 2011.

David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.

Jim Hefferon. Linear Algebra. http://joshua.smcvt.edu/linearalgebra, 2012.

David Joyner and William Stein. Open source mathematical software. *Notices of the AMS*, page 1279, November 2007.

Python Team. Floating point arithmetic: issues and limitations, 2012a. URL http://docs.python.org/2/tutorial/floatingpoint.html. [Online; accessed 17-Dec-2012].

Python Team. The python tutorial, 2012b. URL http://docs.python.org/2/tutorial/index.html. [Online; accessed 17-Dec-2012].

Ron Brandt. *Powerful Learning*. Association for Supervision and Curriculum Development, 1998.

Sage Development Team. Sage tutorial 5.3. http://www.sagemath.org/pdf/SageTutorial.pdf, 2012a.

Sage Development Team. Sage reference manual 5.3. http://www.sagemath.org/pdf/reference.pdf, 2012b.

Shunryu Suzuki. *Zen Mind, Beginners Mind*. Shambhala, 2006.